Statistical Techniques in Robotics (16-831, S21) Lecture #06 (Wednesday, February 17)

Online Convex Optimization (Convexity, FTL)

Lecturer: Kris Kitani *Scribes: Haowen Shi, Fan Jia*

1 Review

1.1 Overview

In the past lectures we have studied a few online learning algorithms including PWEA and Online Linear Classification. In this lecture we study a generalized framework of these online learning algorithms called *Online Convex Optimization* (OCO). We explain why OCO is a generalization of online learning in section 2.2.

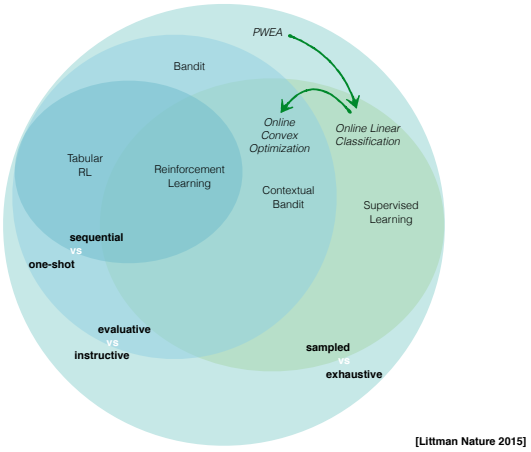

Figure 1: Where we are in the class

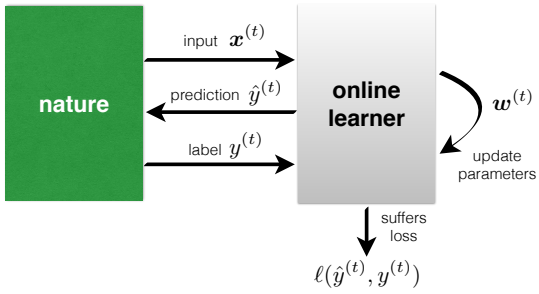

Figure 2: Online learning framework

1.2 Online Learning

In online learning, the prediction model is incrementally improved over multiple iterations. The events that happen in each iteration is illustrated in Fig. 2. At each time step t, the learner receives input $\boldsymbol{x}^{(t)}$, makes a prediction $\hat{y}^{(t)}$ and then receives the true outcome $y^{(t)}$ from the environment. The loss function $l(\hat{y}^{(t)}, y^{(t)})$ is evaluated by the learner to produce a loss which drives the update of model weights $\boldsymbol{w}^{(t)}$.

In previous lectures we have learned about two main classes of online learning:

- Prediction with Expert Advice (PWEA)
  - Greedy / Consistent Algorithm
  - Halving Algorithm

- Weighted Majority Algorithm
- Randomized Weighed Majority Algorithm

- Online Linear Classification
  - Perception Algorithm
  - Winnow Algorithm

Algorithm 1 Perceptron algorithm

$\boldsymbol{w}^{(1)} \leftarrow \mathbf{0}$ ▷ Weight initialization
for $t = 1, \cdots, T$ **do**
  RECEIVE $(\boldsymbol{x}^{(t)} \in \mathbb{R}^N)$ ▷ Receive expert predictions
  $\hat{y}^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t)}, \boldsymbol{x}^{(t)} \rangle\Big)$ ▷ Make learner prediction
  RECEIVE $(y^{(t)} \in \{-1, 1\})$ ▷ Receive actual answer
  $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + y^{(t)} \cdot x^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$ ▷ Weight update
end for

Algorithm 2 Winnow algorithm

$\boldsymbol{w}^{(1)} = \{1, \ldots, 1\}$ ▷ Weight initialization
for $t = 1, \cdots, T$ **do**
  RECEIVE $(\boldsymbol{x}^{(t)} \in \{0, 1\}^N)$
  $\hat{y}^{(t)} = \mathbf{1}[\langle \boldsymbol{w}^{(t)}, \boldsymbol{x}^{(t)} \rangle > N]$ ▷ Make prediction
  RECEIVE $(y^{(t)} \in \{0, 1\})$
  $w_i^{(t+1)} = w_i^{(t)}(1+\beta)^{(y^{(t)} - \hat{y}^{(t)}) \cdot x_i^{(t)}}$ ▷ Weight update
end for

2 Online Convex Optimization

2.1 The Optimization Problem

An optimization problem is the process of finding the best solution among all feasible solutions. Given a function of variables $f(\boldsymbol{x})$, we want to find the set of variables $\boldsymbol{x}$ such that the function produces the optimal (minimum) value $\hat{\boldsymbol{x}}$.

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

In real life, most optimization problems are also *constrained* to meet a set of requirements, these constraints can fall in the following categories:

$$\begin{aligned} \text{Inequality Constraint:} \quad & g(\boldsymbol{x}) \leq 0 \\ \text{Equality Constraint:} \quad & h(\boldsymbol{x}) = 0 \\ \text{Domain Constraint:} \quad & \boldsymbol{x} \in \mathbb{S} \end{aligned}$$

To solve the optimization problem, there are three high level approaches:

1. Analytic solution

   In some optimization problems we have a well defined objective function f so we can find its minima by evaluating its gradient ∇f and setting it to zero. Analytic solution is fast and global for convex objective functions but it is not always possible because the objective function might be too complex to evaluate or sometimes hidden to us.

2. Brute force search

   Brute force search works well in small domain sizes and always obtains globally optimal solutions. The computation soon becomes intractable as the domain size grows so it's not always feasible.

3. Numerical methods

   Solving optimization problems using numerical methods is very popular in the learning community because the computation cost scales well as problem difficulty grows. However, the trade off is numerical solutions are not always globally optimal because they can converge to local minima depending on the initialization.

2.2 Relationship with Online Learning

Online Convex Optimization is a more generalized framework for online learning. In OCO, the learner does not make any observations of the environment, it just sends a set of parameters $\boldsymbol{w}^{(t)}$ to the environment and gets a loss function $l^{(t)}$ back, as shown in Fig. 3 and Algorithm 3. The reason we say OCO is a higher level framework is that we can easily use OCO to solve online learning problems by replacing the "nature" in Fig. 3 with a online learning model. As illustrated in Fig. 4, the online learner in red is calling the online convex optimizer in gray for the prediction step to update its weights.

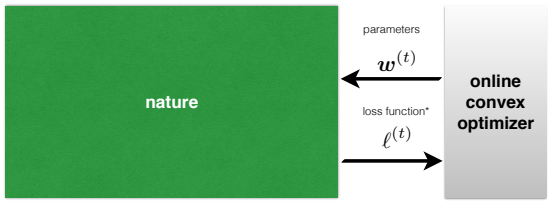

Figure 3: OCO framework

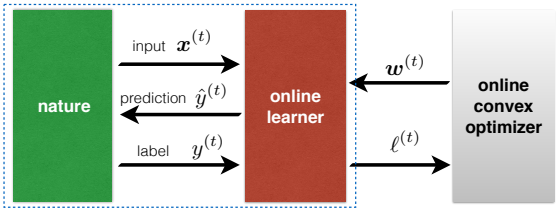

Figure 4: OCO is a generalization

Algorithm 3 Online Convex Optimization

function OnlineConvOpt(Convex set $\mathcal{S}$)
  for $t = 1, 2, \cdots, T$ **do**
    Predict ($\boldsymbol{w}^{(t)} \in \mathcal{S}$) ▷ Solution space must be a convex set
    Receive ($f_t : \mathcal{S} \to \mathbb{R}$) ▷ Loss function must be convex
    Compute Loss ($f_t(\boldsymbol{w}^{(t)})$)
  end for
end function

2.3 Convex Optimization

Generally if a optimization problem is convex, there will be a global optima which can be found faster than non-convex problems.

2.3.1 Convex Set

A set $\mathcal{S}$ is convex if for all $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{S}$, and for all $\alpha \in [0, 1]$, the following holds:

$$\alpha \boldsymbol{w} + (1 - \alpha)\boldsymbol{v} \in \mathcal{S}$$

Geometrically, for any pair of points $\boldsymbol{w}$ and $\boldsymbol{v}$ in $\mathcal{S}$, all the points that lie on the line segment between $\boldsymbol{w}$ and $\boldsymbol{v}$ should also be within the set $\mathcal{S}$, as shown in Fig. 5.

2.3.2 Convex Function

A function $f : \mathcal{S} \to \mathbb{R}$ is convex if for all $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{S}$, and for all $\alpha \in [0, 1]$, the following holds:

$$f(\alpha \boldsymbol{w} + (1 - \alpha)\boldsymbol{v}) \leq \alpha f(\boldsymbol{w}) + (1 - \alpha) f(\boldsymbol{v})$$

Geometrically, this means for any two points on the function, the secant line between two points always lies above the curve segment connecting the two points. Shown in Fig. 6, a function is convex if the green secant line is always higher than the blue curve segment between the point pair for all pair of points in the function.

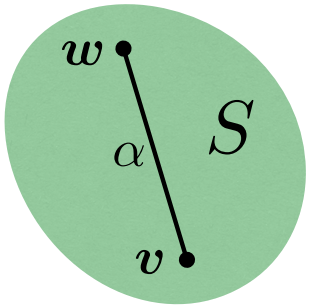

Figure 5: Convex set visualization

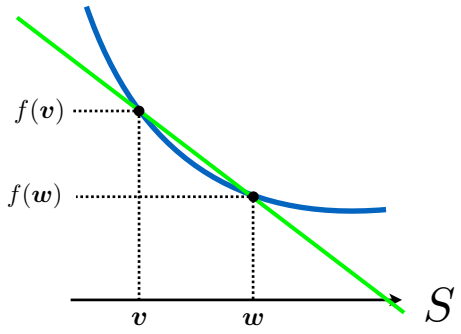

Figure 6: Convex function visualization

Convex optimization requires a convex-Lipschitz-bounded (or convex-smooth-bounded) function and a convex solution space. In the next section, we discuss what is *Lipschitz Continuity*.

2.3.3 Lipschitz Continuity

A function $f(\cdot)$ is called L-Lipschitz smooth over a set $\mathcal{S}$ with respect to a norm $||\cdot||$ if for all $\boldsymbol{u}, \boldsymbol{w} \in \mathcal{S}$, we have:

$$|f(u) - f(w)| \leq L||\boldsymbol{u} - \boldsymbol{w}|| \tag{1}$$

The inequality (1) describes that the rate of change of function f has to be upper bounded by $L||\boldsymbol{u} - \boldsymbol{w}||$ where L is a Lipschitz constant, which limits how fast the function can change. Fig. 7 shows a good visualization [2] of this upper bound: for a Lipschitz-continuous function there exists a double cone (white) that could be moved along the curve without ever intersecting with the curve itself.

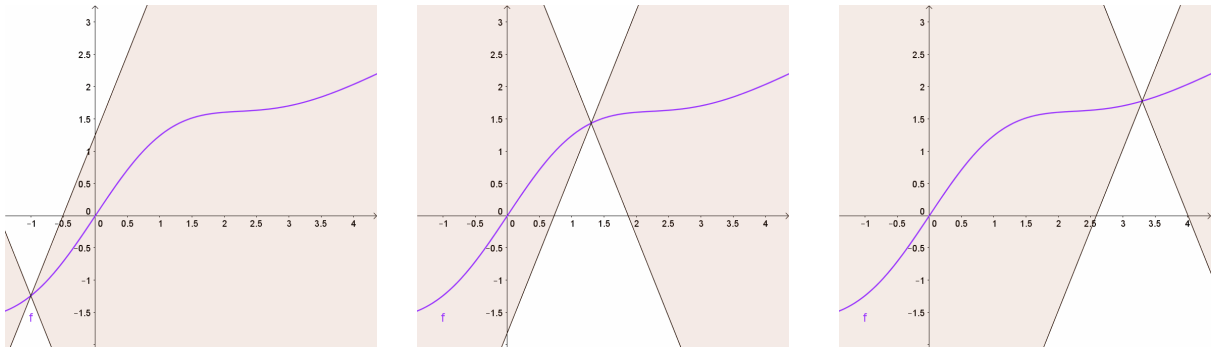

Figure 7: Lipschitz continuity visualization [2]

Note that the Lipschitz continuity has more to do with the analysis of performance guarantee of the algorithm than whether the optimization will work or not. The Lipschitz constant is a mathematical tool for algorithmic analysis instead of a hyperparameter.

2.3.4 Convexification

A loss function is easier to solve when it is convex. If a loss function is not convex, we can use some convexification tricks to make it so. In this section we discuss two methods to convert non-convex loss functions to convex ones.

a. Convexification by randomization
Recall the weighted majority algorithm we studied before where we make prediction by taking the sign of inner product between observation and weights:

$$\hat{y}^{(t)} = \text{sign}\left\langle \boldsymbol{x}^{(t)}, \boldsymbol{w}^{(t-1)} \right\rangle$$

We define the loss function as:

$$l^{(t)} = \mathbf{1}[\hat{y}^{(t)} \neq y^{(t)}]$$

This zero-one loss function is non-convex. Fig. 8, shows this with an example pair of points forming a segment that passes beneath the function.

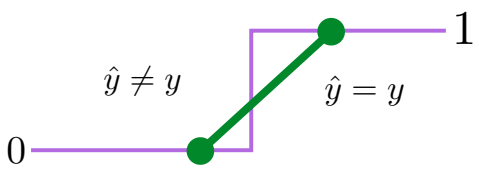

Figure 8: Zero-one loss function is non-convex

Recall that in randomized weighed majority algorithm, we changed the prediction rule to randomly sample an expert's advice from a multinomial distribution.

$$i \sim \textsc{Multinomial}(\boldsymbol{w}^{(t-1)}/\boldsymbol{\phi}^{(t-1)})$$

It turns out that based on such random sampling, the loss function is changed to be an expectation over the zero-one loss.

$$l^{(t)} = E_{\boldsymbol{p}}\left[\mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]\right] = \sum_{p_n} p_n \cdot \mathbf{1}[y^{(t)} \neq \hat{y}_n^{(t)}]$$

Now because of the expectation, the loss function becomes linear and could take any values between zero and one. Therefore, by doing randomization, we converted the non-convex zero-one loss function to a convex function.

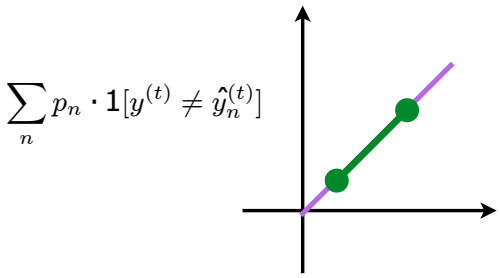

Figure 9: Expectation over zero-one loss function is now convex

b. Convexification by surrogate loss

Instead of dealing with non-convex loss functions, we can design a convex function that upper-bounds the original loss function and solve the optimization problem using convex optimization infrastructure. This idea is called convexification by surrogate loss. We use the weighed majority algorithm again as an example. The original non-convex zero-one loss function is:

$$l^{(t)} = \mathbf{1}[\hat{y}^{(t)} \neq y^{(t)}]$$

We can design a surrogate loss function that upper bounds this loss function to approximate its behavior while making the problem convex. To do so, we need to find a theoretical upper bound for this zero-one loss:

$$\begin{aligned}
l^{(t)} &= \mathbf{1}[\hat{y}^{(t)} \neq y^{(t)}] \\
&= \mathbf{1}[-y^{(t)}\hat{y}^{(t)} > 0] && y \text{ is either -1 or 1} \\
&= \mathbf{1}[-y^{(t)}\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)}\rangle > 0] && \text{substitute (2.3.4)} \\
&= \mathbf{1}[1 - y^{(t)}\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)}\rangle > 1] && \text{shift to the right by 1}
\end{aligned}$$

We can upper bound the original loss function by using a hinge function, as visualized in Fig. 10:

$$\begin{aligned}
l^{(t)} &= \mathbf{1}[1 - y^{(t)}\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)}\rangle > 1] \\
&\leq \max\left[0, 1 - y^{(t)}\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)}\rangle\right] && \text{a convex hinge function } \tilde{l}^{(t)}
\end{aligned}$$

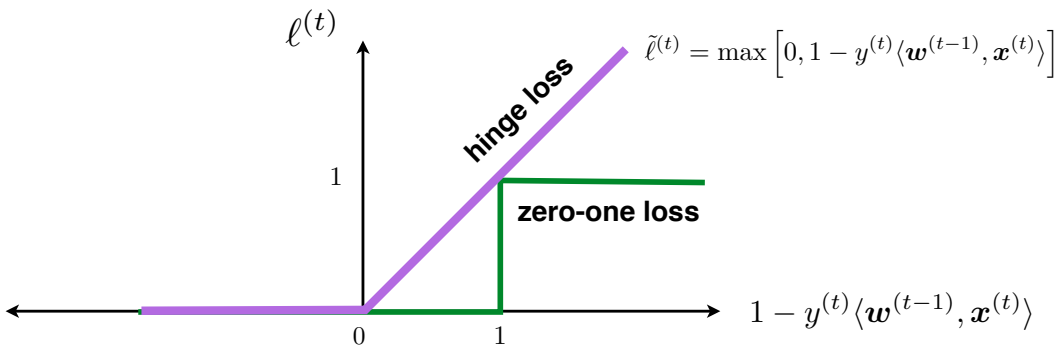

Figure 10: The hinge function acts as a surrogate loss for the zero-one loss function by upper-bounding it

By optimizing the hinge loss $\tilde{l}^{(t)}$, we have a slightly different behavior than the original zero-one loss especially in the [0, 1] domain where the hinge loss produces non-zero penalty while the original loss function does not. This is a trade-off where we make the optimization easier to solve (using an OCO solver) at the cost of an approximation.

3 Follow the Leader

After reviewing the theory part of OCO, we introduce an algorithm called *Follow the Leader* (FTL) known as 'Fictitious Play' (Brown 1951 [1]). The idea is that the learner select the best choice of the game so far. The FTL pseudocode is listed below.

Algorithm 4 Follow the leader algorithm

function FOLLOW THE LEADER
  for $t = 1, 2, \cdots, T$ **do**
    $\boldsymbol{w}^{(t)} = \arg\min_{\boldsymbol{w} \in W} \Sigma_{i=1}^{t-1} f^{(i)}(\boldsymbol{w})$ ▷ Parameter chosen
    RECEIVE $(f^{(t)} : W \to \mathbb{R})$ ▷ Receive loss function
  end for
end function

The prediction rule is that the learner is going to choose the parameter $\boldsymbol{w}$ such that it minimizes the cumulative loss up to now. This rule can be inserted into the online convex optimizer as mentioned in Fig. 3.

Now we derive a general regret bound for the FTL algorithm. First, we use one-step look ahead cheater to get the regret upper bound. Then we apply algebraic tricks to get the final form.

Recall the definition of cumulative regret:

$$R(\boldsymbol{u}) = \sum_t [f^{(t)}(\boldsymbol{w}^{(t)}) - f^{(t)}(\boldsymbol{u})] \tag{2}$$

where $\boldsymbol{u}$ is any hypothesis or parameter. Therefore, the cumulative regret is the sum of all the

differences between the loss of your online convex optimizer and the loss of hypothesis $\boldsymbol{u}$.

Because hypothesis $\boldsymbol{u}$ has infinite possibilities, it is very hard to reason about $\boldsymbol{u}$. Therefore, we introduce "the one-step look ahead cheater" to upper bound the regret so that we can analyze.

$$R(\boldsymbol{u}) = \sum_t [f^{(t)}(\boldsymbol{w}^{(t)}) - f^{(t)}(\boldsymbol{u})] \leq \sum_t [f^{(t)}(\boldsymbol{w}^{(t)}) - f^{(t)}(\boldsymbol{w}^{(t+1)})] \tag{3}$$

where $\boldsymbol{w}^{(t+1)}$ is called "one-step look ahead cheater". It means that the prediction rule at step t that has access to parameters of the next iteration $t+1$. We don't have access to the parameters of the future. However, we can use theoretical values to enable the analysis. The "one-step look ahead cheater" gives us the access to future observations/loss that enables the computation of the best $\boldsymbol{w}$.

To prove the inequality (3), by subtracting $f^{(t)}(\boldsymbol{w}^{(t)})$ from both sides, we only need to prove that:

$$\sum_{t=1}^{T} f^{(t)}(\boldsymbol{w}^{(t+1)}) \leq \sum_{t=1}^{T} f^{(t)}(\boldsymbol{u}) \tag{4}$$

If it holds true, it means that the loss of a series of one step cheaters is not larger than the loss of any other single parameter $\boldsymbol{u}$. Therefore, our claim is:

$$\sum_{t=1}^{T} f^{(t)}(\boldsymbol{w}^{(t+1)}) \leq \sum_{t=1}^{T} f^{(t)}(\boldsymbol{u}) \quad \forall \boldsymbol{u} \tag{5}$$

Proof. By induction method, we first assume our claim is true for $T-1$ and will prove it holds true for T:

$$\sum_{t=1}^{T-1} f^{(t)}(\boldsymbol{w}^{(t+1)}) \leq \sum_{t=1}^{T-1} f^{(t)}(\boldsymbol{u}) \tag{6}$$

Then we add $f^{(T)}(\boldsymbol{w}^{(T+1)})$ to both sides of inequality (6), where $f^{(T)}(\boldsymbol{w}^{(T+1)})$ is the loss of the one step look ahead cheater for the next time step, we can get:

$$\sum_{t=1}^{T} f^{(t)}(\boldsymbol{w}^{(t+1)}) \leq \sum_{t=1}^{T-1} f^{(t)}(\boldsymbol{u}) + f^{(T)}(\boldsymbol{w}^{(T+1)}) \tag{7}$$

Since the inequality (7) holds true for all $\boldsymbol{u}$, if we let $\boldsymbol{u} = \boldsymbol{w}^{(T+1)}$, we can get:

$$\sum_{t=1}^{T} f^{(t)}(\boldsymbol{w}^{(t+1)}) \leq \sum_{t=1}^{T-1} f^{(t)}(\boldsymbol{w}^{(T+1)}) + f^{(T)}(\boldsymbol{w}^{(T+1)}) \tag{8}$$

$$\sum_{t=1}^{T} f^{(t)}(\boldsymbol{w}^{(t+1)}) \leq \sum_{t=1}^{T} f^{(t)}(\boldsymbol{w}^{(T+1)}) \tag{9}$$

It means that the loss of always using one step look ahead cheater is the lower bound of the loss of using a single cheater at the end of the sequence for all time steps. Recall the minimizer by definition,

$$\boldsymbol{w}^{(T+1)} = \arg\min_{\boldsymbol{u}\in S} \sum_{t=1}^{T} f^{(t)}(\boldsymbol{u}) \tag{10}$$

This means the right hand side of the inequality (9) can be upper bounded by $\sum_{t=1}^{T} f^{(t)}(\boldsymbol{u})$. Therefore, we have

$$\sum_{t=1}^{T} f^{(t)}(\boldsymbol{w}^{(t+1)}) \leq \sum_{t=1}^{T} f^{(t)}(\boldsymbol{u}) \tag{11}$$

Therefore, it proves our claim and the inequality (3).

References

[1] U. Berger. Brown's original fictitious play. *Journal of Economic Theory*, 135(1):572–578, 2007.

[2] Wikipedia contributors. Lipschitz continuity — Wikipedia, the free encyclopedia, 2021. [Online; accessed 17-Feb-2021].